AUSTRALIAN OS9 NEWSLETTER

Volume 7 December 1993 Number 11

EDITOR:	Gordon Bentzen	(07) 344-3881
SUB-EDITOR:	Bob Devries	(07) 278-7209
TREASURER:	Jean-Pierre Jacquet	(07) 372-4675
	Fax Messages	(07) 372-8325
LIBRARIAN:	Rod Holden	(07) 200-9870
CONSULTANT:	Don Berrie	(079) 75-3537
SUPPORT:	Brisbane OS9 Users Group	

CONTENTS

Editorial Material:
Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Library Requests:
Rod Holden
53 Haig Road
LOGANLEA Qld 4131

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 7 Number 11

Well it is hard to believe but here we are in December again. Although there doesn't seem to be any scientific evidence to support the theory "that as one gets older the years get shorter", I feel sure that this is happening.

This time of year means different things to different people, the lucky ones will be of on holidays and with time for relaxation or maybe some sun, surf, sailing or whatever. Then of course there are those of us you must slave on in almost the same routine.

Whatever activities you have planned for the Christmas, new-year period we hope that the content of this newsletter will prompt you to begin a new adventure with OS9. That is of course, unless you have a better offer.

We will NOT produce a newsletter in January 1994 as we like to have a break away from the normal routine as well.

IN THIS EDITION

Our librarian and BBS Sysop, Rod Holden, has again submitted an article which we hope will be of interest. This is in fact an article by Zygo Blaxell and provides all you need to make a "DIGITZ", a what? If you have used the "Play" programme from our PD. library then you should take a "read" of Digitz.

BOOT Have you often wondered what is going on while you wait and wait for OS9 to Boot. The Article by Zack Sessions, OS9 Boot, will describe just how busy the bootstrap process is and is well worth reading.

BASIC to Basic09 to C We are often asked if an RSDOS Basic programme can be converted to Basic09 and Bob Devries has presented information on this in the past. The most recent article, October 1993, gave a description of "how to" and now Bob continues with this series and in this edition presents an example of a simple programme which has been converted to Basic09 and also to C.

FAQ 68k The second,and final, part of "Frequently Asked Questions" relating to OS9 68000 is presented in this edition.

OS-9000

You may have wondered about what is happening with OS-9000 since I mentioned that I purchased a copy at the Chicago CoCoFest in May this year. Well although progress has been slow I now hope to spend some more time with it and be able to present a review early next year. One major hold-up was that I didn't have a machine to run it on. I have now purchased a second-hand 80386 40mhz clone, and yes it does run OS-9, OS-9000 that is, and from a users point of view, it has the look and feel of OSK and is "fast".

The big thing that is missing at this point is a "Windows environment" and is on the top of my wish list. The choices for OS-9000 at this time seem to be X-Windows from Microware, or G-Windows from Delmar & Co. Both around US$250 I believe. Until next year, Cheers, Gordon.

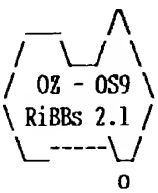
CHRISTMAS
GREETINGS

and
BEST WISHES

for
HAPPINESS

in the
NEW YEAR

FROM - Gordon Bentzen.
Bob Devries.
Jean-Pierre Jacquet.
Rod Holden.
Don Berrie.



The National OS9 Usergroup
(07)-200-9870
300/1200/2400 baud.
20:00 to 22:30 HRS.(AEST)
(8N1)

Co-ordinator: Bob Devries (07)-278-7209
Sysop: Rod Holden

This is (RiBBS).... A Tandy Coco Based BBS program.
This BBS is accessible to Usergroup Members ONLY!
Feel free to look around , and test out the options.

OS9 for Ever !!!!

Hi, this is your Sysop once again letting you know what type of software is available. Here is the document to a program called Digitz for those people who like to record music onto there coco and store it as a file.

Digitz Program
by Zygo Blaxell
for OS-9 Level II on the Color Computer 3

Documentation for versions before version 2.00 deleted. It's irrelevant. Heck, even the hardware for that version was a lousy design.

Version 2.00 documentation

HONEST DISCLAIMER

"Using this software and/or building the hardware as described below may cause damage to your computer, deletion of important files, severe frustration, boredom, Error #248's, fire, floods, war, famine, pestilence, death (well, maybe), Zulu warrior attack, rioting in the streets, assassination attempts, the complete and utter demolition of the solar system, and anything else you may care to think of as well as anything that you didn't think of but which may, at a later date, actually happen, or some similar occurrence which already has happened. However, please note that there may be absolutely no correlation between the explosion into flame of your motherboard and the following of procedures as outlined below. Even if there *IS* one, I accept no liability -- you do *everything* yourself, including taking the flak if things go wrong. While no attempt at completeness or accuracy has been made, you might just find that one or two statements below happen to be relevant anyway." -- The Close Cover Before Striking School of Incomprehensible Legalese (find the nation of Legal on a globe, willya?).

If you should wish to contact the author, you may do so via the Fidonet CoCo or OS9 echo (areatags "COCO" and "OS9"), or you may write to this address...

Zygo Blaxell < -- That works in Fidonet as well!

PO Box 117
Vernon, Ontario, CANADA
K0A 3J0

...where your letter will be ignored as soon as possible.

The hardware design and software are released completely into the public domain, where the untamed masses may do anything they darn well like with it except print it out in purple ink or translate it into Swahili. ;-)

"If you break it, you own both pieces." - Wynn Wagner III.

SOFTWARE

Digitz - OS-9 software sound digitizer. Digitz is capable of digitizing up to two megabytes of sound at user-selectable speeds (max 13-16 KHz). Digitz generates a "Macintosh" format sound file, meaning that the sampled data is stored in straight binary form instead of the oddball format the Amiga uses. It also encodes the sample rate and file format in the first two bytes of the file for compatibility with (Kevin Darling/Brian White / whoever-else-modified-it)'s OS-9 PLAY command. In fact, you need the PLAY command to play back the sound samples once Digitz has generated them, so if you haven't already got a copy of PLAY, then go rummage through your local BBSulary until you find one.

HOW TO START DIGITIZING THINGS

To digitize a sound file sample, get the sound source ready (tape queued up, etc), clean all the crud out of your CoCo's memory, and connect the CoCo to the source (the EARphone jack is best).

Then, get yourself some disk space and type:

DIGITZ filename [speed]

Filename = The file that the sound sample will be sent to.

Speed = Delay between digitizing samples (this is the equivalent PLAY value minus 19). Defaults to 1. Higher numbers give you more recording time per byte, but a loss of fidelity. 1=approx 15.7 KHz.

Digitz will respond with:

Allocating memory.....<lots of dots here>.....
Hit a key when ready to sample...

You may hear a "pop" or "click" at this time -- don't panic, this is normal. Each of the dots represents 8K allocated to Digitz for storage of sound data (Digitz assumes you want to use all of the computer's free memory for sound file sampling). DO NOT KILL DIGITZ! You'll run out of memory REAL fast! When you have everything all set and ready to go, hit any key. Digitz will begin taking the sound sample upon receipt of the keypress and finish when the memory runs out. Later you can "shave" bytes off of the file to remove any unwanted noise at the end of the sound
sample, if so desired.

As soon as sampling is complete, Digitz will deactivate the joystick input (which may cause another "pop" or "click") and dump the sound sample to the specified file, complete with imbedded codes for PLAY. The memory is returned to OS-9 when the file has been written to disk.

ACK! THAT SOUNDS AWFUL!

With a bit of practice, you'll be grinding out pretty good sound file samples. Here are a few things you should know:

1. Volume level. This is, to say the least, important. It's VERY important.

If the source is too quiet, you'll not only get a faint reproduction, but a noisy one as well.

If the source is too loud, you may get "clipping", where an input wave is above 5 or below 0 volts, outside of the range of the CoCo DAC. This manifests itself as either metallic-sounding voices, or loud "pops" and "clicks" (sort of like a scratched record, only this sounds like the record was involved in a cat fight).

If your source has power-level LEDs (those cute bar graph displays that bounce up and down to the music), study the correlation, if any, between these LEDs and undesirable effects on the sample files. They are usually pretty consistent and give even the complete technical klutz something to work with.

2. Harmonics and "swishing". These are a consequence of ANY digital sampling system, but they are most evident when the sample rates are audible.

Harmonics are a form of distortion that occurs when an input wave is an integral multiple of the sample frequency. For instance, if a tone of 10 KHz (a pretty high tone) is input, and the sample rate is exactly 10 KHz, then the input wave will be at the same point in its cycle each time it is sampled. The result of this is that the sampler records the *same voltage* each time through. When this is played back, the tone disappears! By varying the frequency slightly you can get all kinds of strange effects, but if you're trying to sample a piece of music it just sounds like noise (as if there was a tin can on the speaker).

"Swishing" occurs with any digital sampling system, and it is a result of there being a limited number of values to represent an infinite variety of inputs. By decreasing the noise on the input line, increasing the number of values and/or varying the sample rate, the "swishing" can be reduced to below the white noise level, but never truly eliminated. In compact discs, which sample using 16 bits at over 40 KHz, what little swishing there is isn't audible

to human ears (although it should please any passing bats). However, as much as I've tried I cannot make Digitz go faster than 14 KHz without distorting the sample slightly, and we're stuck with a 6-bit DAC on the CoCo, so swish we must.

Version 2.01 documentation

CHANGES

Essentially the only change is in the sampling rate -- described above as "13-15 KHz", you may guess that the rate was somehow uneven. Well, it was. The sample routine has since been changed to a fixed rate of 13.7 KHz, approximately.

NEW PROGRAMS

DLoop is a quick-and-dirty program which will help you determine the precise volume level you need. To run DLOOP simply type "DLOOP" and then adjust your equipment. DLoop will take a two-second sample in its 64k address space, play it back, sleep two ticks (during which you can press BREAK to exit), and repeat indefinitely.

Version 3.00 documentation

The actual software has undergone only cosmetic changes, including response to the "standard" -? help query.

Version 3.01 documentation

HOW IT WORKS (HARDWARE)

This I discovered quite by accident. Apparently, the CoCo seems to have been designed with digitizing in mind! Here's the hookup:

```
CoCo left joystick horizontal input <-----+-> Signal
                                          $ resistor

CoCo +5V <-------------()-----------------+-> Signal
                                          ^ capacitor
```

The actual values for the resistor and capacitor depend on what your input is coming from. 1 uF for the capacitor and 1 megohm for the resistor work for me, but you may have to fiddle with them.

CHANGES

I diddled with the digitizing loop again to deliver some 15455 Hz of sampling speed. If anyone out there can top that, PLEASE let us know!

BBS NEWS

Due to work commitments I can only have the BBS up and running from 2000 - 2130 hrs (AEST). The 68000 software will be available on the BBS from 2 Jan 1994 after I have re-arranged the menus in the BBS. I would like to take this opportunity to wish you and your families a very Merry Christmas and a Happy New Year and look forward to hearing from you in the new year. See you in the bit stream, Happy CoCoing.

Sysop
Rod Holden

How OS9 boots
by Zack Sessions
from an InterNet message, August 1990

We went through the OS9 Level 2 boot procedure a few months ago, but apparently we have some newcomers [I know we do. ED]. Her is exactly what happens, step for step.

Initially some stuff gets executed which is irrelevant to this topic. Then, later, the module INIT is executed. Well, it isn't actually executed, it contains some constants which describe a few things. These are:
1) Startup Module
2) Boot Device
3) Startup Window
4) Name of boot module
In a stock OS9 Level 2 disk from Tandy, the initial values for these are:
1) CC3Go
2) /D0
3) /Term
4) Boot
What this means is that when the module Boot is executing (I think), the data directory is set for /d0 and the execution directory is set for /d0/cmds. Then CC3Go is forked in the /term device. Now on to

CC3Go. CC3Go has a few constants as well. They are:
1) Startup Banner Message
2) Data Directory
3) Execution Directory
4) Startup Program
5) "autostartup" Program
6) Startup Procedure Filename
The initial values in the stock CC3Go module from Tandy are:
1) OS9 Level 2 V02.00.01, etc.
2) /H0
3) /H0/CMDS
4) Shell
5) Autoex
6) STARTUP
CC3Go is the key module for OS9 Level 2 on a Color Computer 3. What it does is this: (this isn't everything, I left a few things out, but this is relevant to this discussion).
1) Writes out banner message to startup window. By default this is /term, and by default /term is a VDG type window. If you patch Init to startup in a window, say /W or /W1, and the window device type is a window device, not a VDG window, then CC3Go performs an implied load of GrfDrv. Since your device is still /d0, then grfdrv MUST be in /d0/cmds with the execution attribute bit set. If not, "OS9 BOOT FAILED". If you use a hard drive and you want to boot up in a window and you want to get Grfdrv from the hard drive, you must also patch Init's Startup Device to either the hard drive or (more commonly) to /dd (if /dd equates to your hard drive, of course).
2) Attempt to do a "chx cmds" command. The Cmds is the later part of the Startup Execution Directory string. If an error occurs, it is ignored. (I'm not really sure why this is even done because it is my impression that by this time the execution directory is already set as /d0/cmds)
3) Attempt to do a "chd /H0" command. The /h0 is the string Data Directory, number 2 above. If an error occurs, skip 4, go to step 5.
4) Attempt to do a "chx /H0/cmds" command. The /H0/cmds is the string Execution Directory, number 3 above. If an error occurs, it is ignored.
5) Attempt to fork a Shell feeding it STARTUP as the parameter. The command Shell is the Startup Program, number 4 above, and STARTUP is the Startup Procedure Filename, number 6 above. If this fails, then "OS9 BOOT FAILED". This is the first time Shell is needed. This implies that you have a device called /h0, and there is a directory called /h0/cmds, then Shell must be there. If you don't then Shell must be in /d0/cmds. Of course, in either location, it must have its execution attribute set on. On success of this fork, wait for the child process to complete.
6) Attempt to fork a child process with AutoEx as the primary module. AutoEx is the "autostartup" program, number 5 above. If this succeeds a wait is performed, and CC3Go does not continue until that program, whatever it may be, issues an F$Exit call. If this fork fails, the error is ignored. The most common use for this "feature" is to automatically start GShell at boot time. The Multi-Vue disk has an image called autoex in its CMDS directory which is nothing more than a copy of the program multistart with its name set as autoex. If you don't believe me, try running an Ident on the two files.
7) Attempt to Chain an immortal Shell in the current window device, normally /Term. Here, Shell is the same Startup Program, number 4 above. To force an immortal shell to be chained, it is passed a parameter string, "i=/1". If this fails, "OS9 BOOT FAILED".

Customising your boot procedure involves patching Init and CC3Go's constants to be what you want. In my case, I have patched Init to come up in window device /W and with a boot device of /dd. I have also patched CC3Go in the following manner:
1) Changed the Banner Message to something like "Property of Zack Sessions", etc.
2) Added code to set the monitor type to RGB, set the mouse to hi-res/right port, and turn off the floppy motors.

Doing these required disassembling CC3Go, making changes to the source, and recompiling with the Level 1 assembler, ASM. If you want more specific instructions, let me know.
Hope this helps someone!
Zack Sessions
Sessions@Sparev.dnet.ge.com (Internet)
!uflorida!ki4pv!macs!stetson!rewop!sencland!sessions (UUCP)

Converting BASIC programmes
by Bob Devries

Continuing with my series on conversion of RSDOS BASIC programmes to run under OS9, I have the following programmes for you to compare. First, a programme, supplied by one of our members, in BASIC, which calculates parallel or series resistor networks. The programme is reproduced exactly as supplied, and is a good example of normal, but less than ideal, BASIC programming. The sample has a number of errors, which I have removed in the following Basic09 and C versions. You'll notice in particular, that the sample uses GOTO, RUN, and DIMensions variable arrays (un-necessarily) after finding the required size. I say un-necessarily, because an array was not necessary at all! Also, there was not way out of the programme except for pressing the BREAK key! Also, there are two lines that are never reached! Here is the original code:

```
5 CLS
10 PRINT:PRINT STRING$(64,"*")
15 PRINT" THIS PROGRAMME WILL CALCULATE TOTAL RESISTANCE OF RESISTORS IN SERIES
OR PARALLEL"
20 PRINT:PRINT" ARE RESISTORS IN SERIES (S)"
25 PRINT" OR ARE THEY IN PARALLEL (P)"
30 PRINT
40 PRINT STRING$(64,"*")
45 PRINT "ENTER (S) OR (P)"
50 INPUT A$
60 IF A$="S" THEN GOTO 100
70 IF A$="P" THEN GOTO 400
75 IF A$<>"" THEN 10
80 GOTO 30
100 CLS
105 PRINT STRING$(32,"*")
110 PRINT "HOW MANY RESISTORS ARE IN SERIES ";
130 INPUT N
135 IF N=1 OR N<1 THEN 110
150 PRINT
170 PRINT "INPUT VALUES IN OHMS"
190 LET R=0
200 DIM V(N)
230 FOR J=1 TO N
240 PRINT "R";J;"=";
250 INPUT V(N)
270 LET R=R+V(N)
280 NEXT J
290 PRINT
300 PRINT " R = ";R"OHMS"
310 PRINT "HIT ENTER FOR NEXT CALCULATION"
320 INPUT Z$:RUN
330 CLS
340 GOTO 10
400 CLS
405 PRINT STRING$(32,"*")
410 PRINT "HOW MANY RESISTORS IN PARALLEL";
420 INPUT N
425 IF N=1 OR N<1 THEN 410
450 PRINT
460 LET RR=0
470 DIM A(N)
```

```
480 PRINT "INPUT EACH RESISTANCE IN OHMS"
490 FOR K=1 TO N
510 PRINT "R";K;"=";
520 INPUT A(K)
540 LET RR=RR+(1/A(K))
550 NEXT K
560 PRINT
580 TR=1/RR
590 IF TR<1E4 THEN 600 ELSE 610
600 PRINT USING "TOTAL RESISTANCE = ####.## OHMS";TR:GOTO 310
610 PRINT "TOTAL RESISTANCE =";TR;"OHMS":GOTO310
```

OK, here's the Basic09 version. Notice the total lack of LINE NUMBERS. Check out the differences between the programmes, and the similarities! Please note that the numbers down the left side are NOT line numbers, but memory locations supplied by Basic09 for reference should there be errors, you do NOT type these into the Basic09 editor!

Note that BASIC's CLS command is not available in Basic09, and is replaced with PRINT CHR$(12) which issues a formfeed on a CoCo. This is not quite kosher in OS9 terms, since the clear screen character COULD be anything, but it will do for this example. Also, no STRING$ function exists, so a FOR-NEXT loop was used. In checking user response to questions, I made sure I checked for upper AND lower case characters, since I don't know which is being used. This is not as important in BASIC, since its default is UPPERCASE. With Basic09 you MUST DIMension ALL variables if you want correct results. If, when you use a numeric variable, you don't DIMension it first, the variable TYPE used will be REAL (floating point). So what? Well, it takes up more room, and also when printed, is followed by a period, like this '512.', to show it is a REAL number. If you use string variable without DIMensioning, you MUST use the '$' (as in a$) and the default length is 32 characters, and cannot be changed without using DIM.

```
PROCEDURE resistor
 0000       (* dimension variables *)
 0019       DIM x:INTEGER
 0020       DIM a$:STRING[1]
 002C       DIM n:INTEGER
 0033       DIM j:INTEGER
 003A       DIM r:REAL
 0041       DIM rtot:REAL
 0048       DIM z$:STRING[1]
 0054
 0055       (* main body of programme *)
 0071       LOOP  \(* don't use line numbers *)
 008F         rtot=0
 0097         z$=" "
 009F         PRINT CHR$(12)
 00A4         FOR x=1 TO 64 \ PRINT "*";  \NEXT x \ PRINT
 00C7         PRINT " This programme will calculate the total resistance"
 00FE         PRINT " of resistors in series or parallel"
 0125         PRINT  \ PRINT
 0129         PRINT " Are resistors in series (type S)"
 014E         PRINT " or are they in parallel (type P)"
 0173         PRINT
 0175         FOR x=1 TO 64 \ PRINT "*";  \NEXT x \ PRINT
 0198         PRINT " Enter S or P ";
 01AB         INPUT a$
 01B0         IF a$="s" OR a$="S" THEN
 01C5           PRINT CHR$(12)
 01CA           FOR x=1 TO 34 \ PRINT "*";  \NEXT x \ PRINT
```

```
01ED          PRINT "How many resistors are in series ";
0213          INPUT n
0218          FOR j=1 TO n
0229            PRINT "R"; j; "=";
0237            INPUT r
023C            rtot=rtot+r
0248          NEXT j
0253          PRINT  \ PRINT " R = "; rtot; " Ohms."
026B          PRINT  \ PRINT " Hit ENTER for next calculation"
0290          PRINT " or Q to quit."
02A2          WHILE z$<>"" DO
02AE            INPUT z$
02B3          EXITIF z$="q" OR z$="Q" THEN
02C8          ENDEXIT
02CC          ENDWHILE
02D0        ENDIF
02D2        IF a$="p" OR a$="P" THEN
02E7          PRINT CHR$(12)
02EC          FOR x=1 TO 36 \ PRINT "*";  \NEXT x \ PRINT
030F          PRINT "How many resistors are in parallel ";
0337          INPUT n
033C          FOR j=1 TO n
034D            PRINT "R"; j; "=";
035B            INPUT r
0360            rtot=rtot+1/r
0370          NEXT j
037B          rtot=1/rtot
0387          PRINT  \ PRINT " R = "; rtot; " Ohms."
039F          PRINT  \ PRINT " Hit ENTER for next calculation"
03C4          PRINT " or Q to quit."
03D6          WHILE z$<>"" DO
03E2            INPUT z$
03E7          EXITIF z$="q" OR z$="Q" THEN
03FC          ENDEXIT
0400          ENDWHILE
0404        ENDIF
0406      EXITIF z$="q" OR z$="Q" THEN
041B      ENDEXIT
041F      ENDLOOP
0423      END
```

OK, then, here's the C version:

```
/* resistor.c */
/* calculate series or parallel combinations */
#include <stdio.h>

main()
{
    int x,n,j;
    double r,rtot;
    double atof();
    int z,a;
    char ns[20];
    char rs[20];
```

```
pffinit(); /* these two functions set up printf() */
pflinit(); /* to print floats and longs          */

setbuf(stdout,NULL);
setbuf(stdin,NULL);

for (;;) {
rtot = 0.0;
putchar('\x0C'); /* output a formfeed to cls  */
     for (x=1;x<65;x++)
          putchar('*');
     putchar('\n');

     printf(" This programme will calculate the total resistance\n");
     printf(" of resistors in series or parallel\n\n");
     printf(" Are resistors in series (type S)\n");
     printf(" or are they in parallel (type P)\n\n");

     for (x=1;x<65;x++)
          putchar('*');
     putchar('\n');
     printf(" Enter S or P ");
     a = getchar();

     if ((a == 'S')    (a == 's')) {
          putchar('\x0C');
          for (x=1;x<35;x++)
               putchar('*');
          putchar('\n');

          printf("How many resistors in series ");
          gets(ns);
          n = atoi(ns);
          putchar('\n');

          for (j=1;j<n+1;j++) {
               printf("R%d = ",j);
               gets(rs);
               putchar('\n');
               r = atof(rs);
               rtot += r;
          }
          printf("\n R = %f Ohms.\n",rtot);
          printf("\n Hit ENTER for next calculation\n");
          printf(" or Q to quit. ");
          do {
               z = getchar();
               if ((z == 'q')    (z == 'Q'))
                    exit(0);
          } while (z != '\n');
     }
     if ((a == 'P')    (a == 'p')) {
          putchar('\x0C');
          for (x=1;x<36;x++)
               putchar('*');
```

```
                putchar('\n');
                printf("How many resistors in parallel ");
                gets(ns);
                putchar('\n');
                n = atoi(ns);

                for (j=1;j<n+1;j++) {
                     printf("R%d = ",j);
                     gets(rs);
                     putchar('\n');
                     r = atof(rs);
                     rtot += (1.0/r);
                }
                rtot = 1.0 / rtot;
                printf("\n R = %f Ohms.\n",rtot);
                printf("\n Hit ENTER for next calculation\n");
                printf(" or Q to quit.");
                do {
                     z = getchar();
                     if ((z == 'q') || (z == 'Q'))
                          exit(0);
                } while (z != '\n');
          }
     }
}

/* EOF */
```

OK, that's it for now. If you're like me, you'll learn from other people's code, and be able to apply that to what you need for yourself.

FREQUENTLY ASKED QUESTIONS OSK

Continued from last month ..

[S-Record Format]

Chaplin@keinstr.uucp (Roger Chaplin) reposted an article written by mcdchg!motmpl!ron (Ron Widell) that explained how Motorola S-Records are formatted. This comes from a unix man page. No mention of which version of Unix is specified. This section of the FAQ is a bit long. An anonymous ftp archive is currently being sought. When one is found, the section will be placed in the archive.

SREC(4) UNIX 5.0 (03/21/84)

An S-record file consists of a sequence of specially formatted ASCII character strings. An S-record will be less than or equal to 78 bytes in length.

The order of S-records within a file is of no significance and no particular order may be assumed.

The general format of an S-record follow:

```
+-------------------------------------------------+
! type ! count ! address  !   data      ! checksum !
!      !       !          !             !          !
---------------------------------------------------+
```

type A char[2] field. These characters describe the type of record - (S0, S1, S2, S3, S5, S7, S8, or S9).

count A char[2] field. These characters when paired and interpreted as a hexadecimal value, display the count of remaining character pairs in the record.

address A char[4,6, or 8] field. These characters grouped and interpreted as a hexadecimal value display the address at which the data field is to be loaded into memory. The length of the field depends on the number of bytes necessary to hold the address. A 2-byte address uses 4 characters, a 3-byte address uses 6 characters, and a 4-byte address uses 8 characters.

data A char [0-64] field. These characters when paired and interpreted as hexadecimal values represent the memory loadable data or descriptive information.

checksum A char[2] field. These characters when paired and interpreted as a hexadecimal value display the least significant byte of the ones complement of the sum of the byte values represented by the pairs of characters making up the count, the address, and the data fields.

Each record is terminated with a line feed. If any additional or different record terminator(s) or delay characters are needed during transmission to the target system it is the responsibility of the transmitting program to provide them.

S0 Record The type of record is 'S0' (0x5330).

The address field is unused and will be filled with zeros (0x0000). The header information within the data field is divided into the following subfields.

mname is char[20] and is the module name.
ver is char[2] and is the version number.
rev is char[2] and is the revision number.
description is char[0-36] and is a text comment.

Each of the subfields is composed of ASCII bytes whose associated characters, when paired, represent one byte hexadecimal values in the case of the version and revision numbers, or represent the hexadecimal values of the ASCII characters comprising the module name and description.

S1 Record
The type of record field is 'S1' (0x5331). The address field is interpreted as a 2-byte address. The data field is composed of memory loadable data

S2 Record
The type of record field is 'S2' (0x5332). The address field is interpreted as a 3-byte address. The data field is composed of memory loadable data

S3 Record
The type of record field is 'S3' (0x5333). The address field is interpreted as a 4-byte address. The data field is composed of memory loadable data

S5 Record
The type of record field is 'S5' (0x5335). The address field is interpreted as a 2-byte value and contains the count of S1, S2, and S3 records previously transmitted. There is no data field.

S7 Record
The type of record field is 'S7' (0x5337)
The address field contains the starting execution address and is interpreted as 4-byte address.
There is no data field.

S8 Record
The type of record field is 'S8' (0x5338)
The address field contains the starting execution address and is interpreted as 3-byte address.
There is no data field.

S9 Record
The type of record field is 'S9' (0x5339). The address field contains the starting execution address and is interpreted as 2-byte address.
There is no data field.

EXAMPLE

Shown below is a typical S-record format file.

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S113001000020008000826290018538123410018l3
S11300204lE900084E42234300182342000824A952
S107003000144ED492
S5030004F8
S9030000FC
```

The file consists of one S0 record, four S1 records, one S5 record and an S9 record.

The S0 record is comprised as follows:

S0	S-record type S0, indicating it is a header record.
06	Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.
00 00	Four character 2-byte address field, zeroes in this example.
48	ASCII H, D, and R - "HDR".
1B	The checksum.

The first S1 record is comprised as follows:

S1	S-record type S1, indicating it is a data record to be loaded at a 2-byte address.
13	Hexadecimal 13 (decimal 19), indicating that nineteen character pairs, representing a 2 byte address, 16 bytes of binary data, and a 1 byte checksum, follow
00 00	Four character 2-byte address field; hexidecimal address 0x0000, where the data which follows is to be loaded.
28 5F 24 5F 22 12 22 6A 00 04 24 29 00 08 23 7C	Sixteen character pairs representing the actual binary data.
2A	The checksum.

The second and third S1 records each contain 0x13 (19) character pairs and are ended with checksums of 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92

The S5 record is comprised as follows:

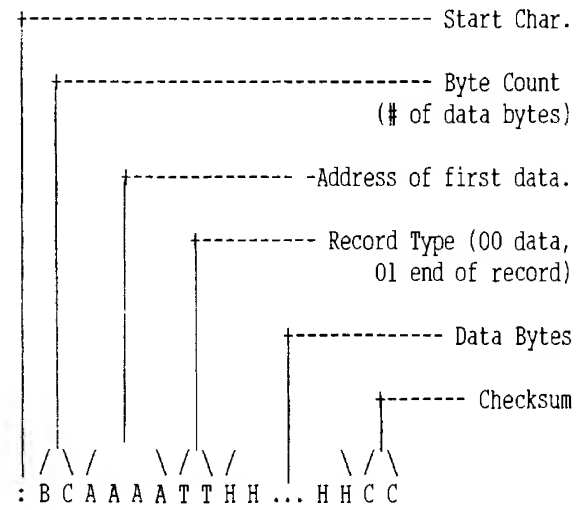
S5	S-record type S5, indicating it is a count record indicating the number of S1 records
03	Hexadecimal 03 (decimal 3) indicating that three character pairs follow.
00 04	Hexadecimal 0004 (decimal 4), indicating that there are four data records previous to this record.
F8	The checksum.

The S9 record is comprised as follows:

S9	S-record type S9, indicating it is a termination record.
03	Hexadecimal 03 (decimal 3), indicating that three character pairs follow.
00 00	The address field, hexadecimal 0 (decimal 0) indicating the starting execution address.
FC	The checksum.

[Intel Hex ASCII Format]

Intel HEX-ASCII format takes the form:



An examples:

```
:10000000DB00E60F5F1600211100197ED300C3004C
:1000100000000010103030707OF0F1F1F3F3F7F7FF2
:01002000FFE0
:00000001FF
```

This information comes from _Microprocessors and Programmed Logic_, Second Edition, Kenneth L. Short, 1987, Prentice-Hall, ISBN 0-13-580606-2.

Provisions have been made for data spaces larger than 64 kBytes. The above reference does not discuss them. I suspect there is a start of segment type record, but I do not know how it is implemented.

[Contributors]

Many people have contributed to this list. Below is a list of people who have helped by either their direct input or through their postings to comp.sys.m68k. I can't list everyone, but I have tried to include many.

walvdrk_r@research.ptt.nl (Kees van der Wal)
byron@cc.gatech.edu (Byron A Jeff)
mcdchg!motmpl!ron (Ron Widell)
rrt@mpd.tandem.com (Bob Teisberg)
benstn@olivetti.nl (Ben Stuyts)
csuley@cs.cornell.edu (Christopher Suley)
idr@mailhost.cs.pdx.edu (Ian D Romanick)
wayne@netcom.com (wayne t. watson)
kevin@mml001.chi.il.us (Kevin J Pease)
gt@prosun.first.gmd.de (Gerd Truschinski)